

Dependency Parsing

COM4513/6513 Natural Language Processing

Nikos Aletras

`n.aletras@sheffield.ac.uk`

`@nikalettras`

Computer Science Department

Week 5

Spring 2020



The
University
Of
Sheffield.

In previous lectures...

- **Text Classification:** Given an instance x (e.g. document), predict a label $y \in \mathcal{Y}$
- **Tasks:** sentiment analysis, topic classification, etc.
- **Algorithm:** Logistic Regression

In previous lectures...

- **Sequence labelling:** Given a sequence of words $\mathbf{x} = [x_1, \dots, x_N]$, predict a sequence of labels $\mathbf{y} \in \mathcal{Y}^N$
- **Tasks:** part of speech tagging, named entity recognition, etc.
- **Algorithms:** Hidden Markov Models, Conditional Random Fields

In this lecture...

- Model richer linguistic representations: **graphs**
- **Dependency parses:** Graphs representing syntactic relations between words in a sentence

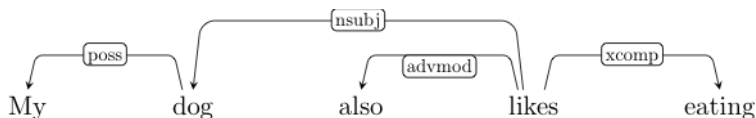
In this lecture...

- Model richer linguistic representations: **graphs**
- **Dependency parses:** Graphs representing syntactic relations between words in a sentence
- Two approaches:
 - Graph-based Dependency Parsing
 - Transition-based Dependency Parsing

Dependency Parsing: Applications

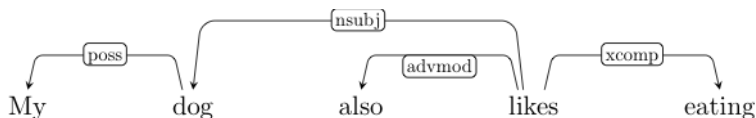
- **Relation extraction**, e.g. identify entity pairs (AM, Arctic Monkeys), (Abbey Road, Beatles), (Different Class, Pulp) with the relation `music_album_by`
- **Question answering**
- **Sentiment analysis**

What is a Dependency Parse?



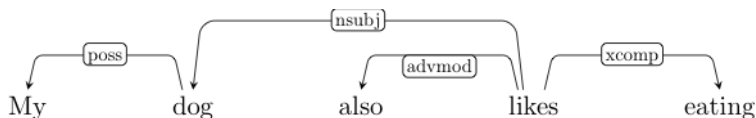
- **Dependency parse (or tree):** Graph representing syntactic relations between words in a sentence

What is a Dependency Parse?



- **Dependency parse (or tree):** Graph representing syntactic relations between words in a sentence
- **Nodes (or vertices):** Words in a sentence
- **Edges (or arcs):** Syntactic relations between words, e.g. dog is the subject (nsbj) of likes ([list of standard dependency relations](#))

What is a Dependency Parse?



- **Dependency parse (or tree):** Graph representing syntactic relations between words in a sentence
- **Nodes (or vertices):** Words in a sentence
- **Edges (or arcs):** Syntactic relations between words, e.g. dog is the subject (nsbj) of likes ([list of standard dependency relations](#))
- **Dependency Parsing:** Automatically identify the syntactic relations between words in a sentence

Graph constraints

- **Connected:** every word can be reached from any other word ignoring edge directionality

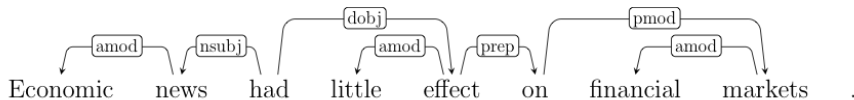
Graph constraints

- **Connected:** every word can be reached from any other word ignoring edge directionality
- **Acyclic:** can't re-visit the same word on a directed path

Graph constraints

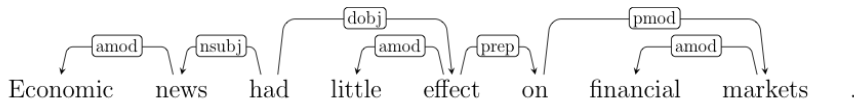
- **Connected:** every word can be reached from any other word ignoring edge directionality
- **Acyclic:** can't re-visit the same word on a directed path
- **Single-Head:** every word can have only one head

Well-formed Dependency Parse?



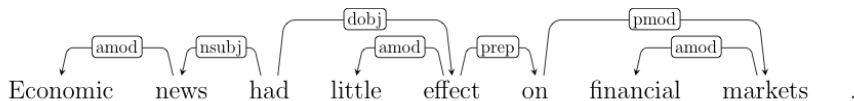
■ Connected?

Well-formed Dependency Parse?



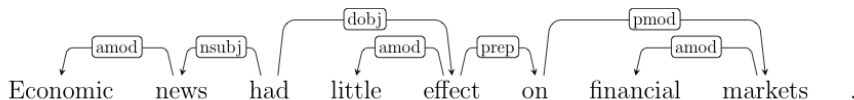
■ Connected? NO

Well-formed Dependency Parse?



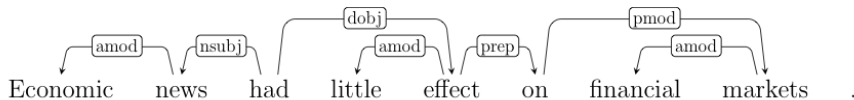
- Connected? NO
- Acyclic?

Well-formed Dependency Parse?



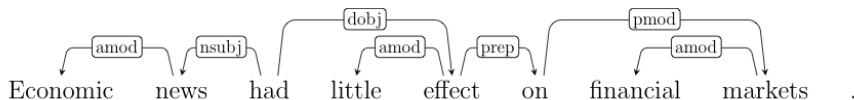
- Connected? NO
- Acyclic? YES

Well-formed Dependency Parse?



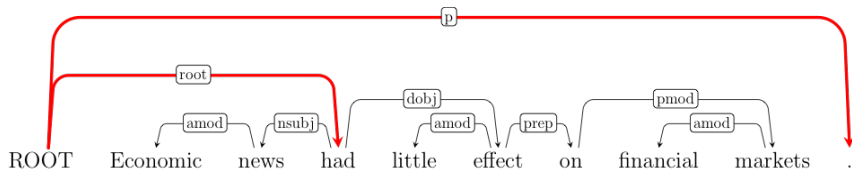
- Connected? NO
- Acyclic? YES
- Single-headed?

Well-formed Dependency Parse?



- Connected? NO
- Acyclic? YES
- Single-headed? YES
- Solution?

Well-formed Dependency Parse



Add a **special root node** with edges to any nodes without heads (main verb and punctuation).

Dependency Parsing: Problem setup

Training data is pairs of word sequences (sentences) and dependency trees:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\text{graph } G_x = (V_x, A_x)$$

$$\text{vertices } V_x = \{0, 1, \dots, N\}$$

$$\text{edges } A_x = \{(i, j, k) | i, j \in V, k \in L(\text{labels})\}$$

Dependency Parsing: Problem setup

Training data is pairs of word sequences (sentences) and dependency trees:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\text{graph } G_x = (V_x, A_x)$$

$$\text{vertices } V_x = \{0, 1, \dots, N\}$$

$$\text{edges } A_x = \{(i, j, k) | i, j \in V, k \in L(\text{labels})\}$$

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, \mathbf{x})$$

Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, \mathbf{x})$$

where the G_x is a well-formed dependency tree.

- **Can we learn it using what we know so far?**

Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, \mathbf{x})$$

where the G_x is a well-formed dependency tree.

- **Can we learn it using what we know so far?** Enumeration over all possible graphs will be expensive.

Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, x)$$

where the G_x is a well-formed dependency tree.

- **Can we learn it using what we know so far?** Enumeration over all possible graphs will be expensive.
- **How about a classifier that predicts each edge?**

Learning a dependency parser

We want to learn a model to predict the best graph:

$$\hat{G}_x = \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, \mathbf{x})$$

where the G_x is a well-formed dependency tree.

- **Can we learn it using what we know so far?** Enumeration over all possible graphs will be expensive.
- **How about a classifier that predicts each edge?** Maybe. But predicting an edge makes some edges invalid due to the acyclic and single-head constraints.

Maximum Spanning Tree

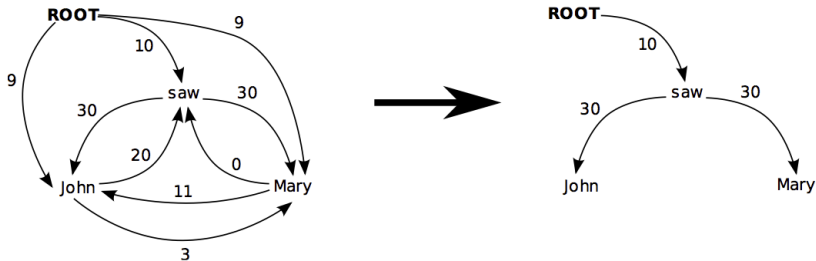
- **Spanning Tree:** In graph theory, a spanning tree T of an undirected graph G is a subgraph that includes all of the vertices of G , with the minimum possible number of edges.

Maximum Spanning Tree

- **Spanning Tree:** In graph theory, a spanning tree T of an undirected graph G is a subgraph that includes all of the vertices of G , with the minimum possible number of edges.
- **Tree:** In computer science, a tree is a widely used data structure (Abstract Data Type) that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes.

Maximum Spanning Tree

Score all edges, but keep only the max spanning tree using Chu-Liu-Edmonds algorithm, a modification to Kruskal's algorithm for extracting Maximum Spanning Trees.



Exact solution in $O(N^2)$ time using Chu-Liu-Edmonds.

Kruskal's algorithm

Input *scored edges E*

sort E by cost (opposit of score)

$G = \{\}$

while *G not spanning* **do:**

pop the next edge e

if *connecting different trees :*

add e to G

Return G

Graph-based Dependency Parsing

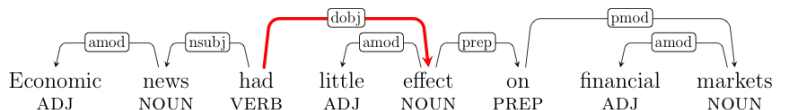
Decompose the graph score into arc scores:

$$\begin{aligned}\hat{G}_x &= \arg \max_{G_x \in \mathcal{G}_x} \text{score}(G_x, \mathbf{x}) \\ &= \arg \max_{G_x \in \mathcal{G}_x} \mathbf{w} \cdot \Phi(G_x, \mathbf{x}) \quad (\text{linear model}) \\ &= \arg \max_{G_x \in \mathcal{G}_x} \sum_{(i,j,l) \in A_x} \mathbf{w} \cdot \phi((i,j,l), \mathbf{x}) \quad (\text{arc-factored})\end{aligned}$$

Can learn the weights with a Conditional Random Field!

Feature Representation

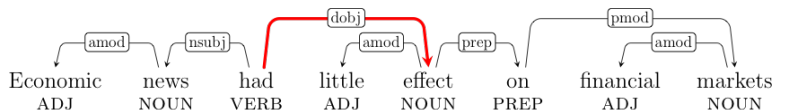
What should $\phi((head, dependent, label), \mathbf{x})$ be?



- **unigram:** head=had, head=VERB

Feature Representation

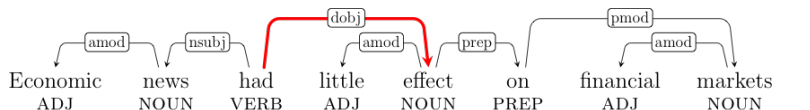
What should $\phi((head, dependent, label), \mathbf{x})$ be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect

Feature Representation

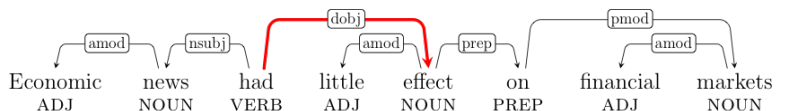
What should $\phi((head, dependent, label), \mathbf{x})$ be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect
- head=VERB & dependent=NOUN & between=ADJ

Feature Representation

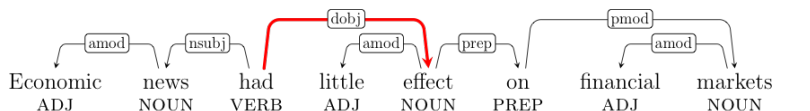
What should $\phi((head, dependent, label), \mathbf{x})$ be?



- **unigram:** head=had, head=VERB
- **bigram:** head=had & dependent=effect
- head=VERB & dependent=NOUN & between=ADJ
- head=had & label=dobj & other-label=nsubj

Feature Representation

What should $\phi((head, dependent, label), \mathbf{x})$ be?



- **unigram:** head=had, head=VERB
 - **bigram:** head=had & dependent=effect
 - head=VERB & dependent=NOUN & between=ADJ
 - head=had & label=dobj & other-label=nsubj
- NO!!! Breaks the arc-factored scoring

More Global models

- Even though inference and learning are global, features are localised to arcs.
- **Can we have more global features?**

More Global models

- Even though inference and learning are global, features are localised to arcs.
- **Can we have more global features?** Yes we can! Consider subgraphs spanning a few edges. But inference becomes harder, requiring more complex dynamic programs and clever approximations.
- **Is it worth it?** Syntactic parsing has many applications, thus better compromises between speed and accuracy are always welcome!

Transition-based Dependency Parsing

- Graph-based dependency parsing restricts the features to perform joint inference efficiently.

Transition-based Dependency Parsing

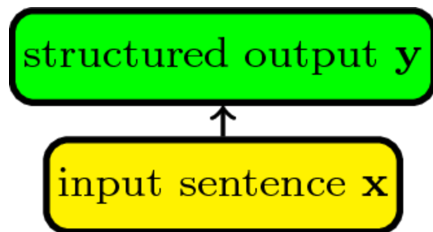
- Graph-based dependency parsing restricts the features to perform joint inference efficiently.
- **Transition-based dependency parsing** trades joint inference for feature flexibility.

Transition-based Dependency Parsing

- Graph-based dependency parsing restricts the features to perform joint inference efficiently.
- **Transition-based dependency parsing** trades joint inference for feature flexibility.
- No more argmax over graphs, just use a classifier with any features we want!

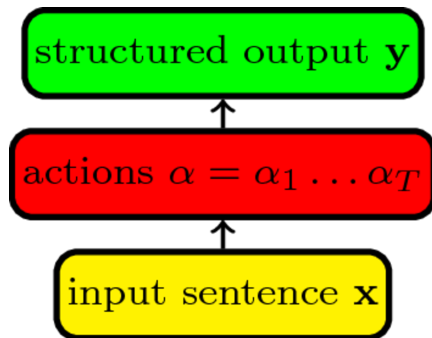
Joint vs incremental prediction

Joint: score (and enumerate) complete outputs (graphs)



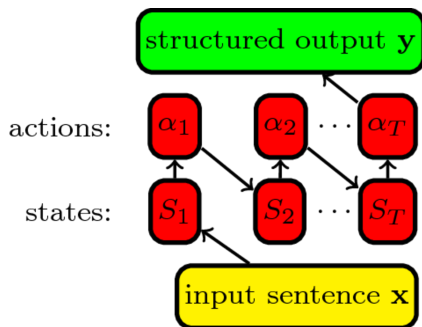
Joint vs incremental prediction

Incremental: predict a sequence of actions (transitions)
constructing the output



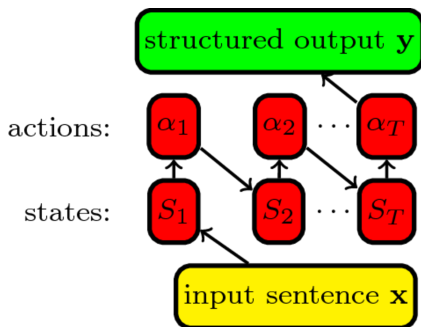
Transition system

The **actions** \mathcal{A} the classifier f can predict and their effect on the **state** which tracks the prediction: $S_{t+1} = S_1(\alpha_1 \dots \alpha_t)$



Transition system

The **actions** \mathcal{A} the classifier f can predict and their effect on the **state** which tracks the prediction: $S_{t+1} = S_1(\alpha_1 \dots \alpha_t)$



What should the actions (transitions) be for dependency parsing?

Transition system setup

- **Input:** Vertices $V_x = \{0, 1, \dots, N\}$ (words sentence \mathbf{x})
- **State** $S = (Stack, B, A)$:
 - Arcs A (dependencies predicted so far)
 - Buffer Buf (words left to process)
 - Stack $Stack$ (last-in, first out memory)
- Initial state: $S_0 = ([], [0, 1, \dots, N], \{\})$
- Final state: $S_{final} = (Stack, [], A)$

Transition system

Shift $(Stack, i|Buf, A) \rightarrow (Stack|i, Buf, A)$: push next word from the buffer (i) to stack

Transition system

Shift $(Stack, i|Buf, A) \rightarrow (Stack|i, Buf, A)$: push next word from the buffer (i) to stack

Reduce $(Stack|i, Buf, A) \rightarrow (Stack, Buf, A)$: pop word top of the stack (i) if it has a head

Transition system

Shift $(Stack, i|Buf, A) \rightarrow (Stack|i, Buf, A)$: push next word from the buffer (i) to stack

Reduce $(Stack|i, Buf, A) \rightarrow (Stack, Buf, A)$: pop word top of the stack (i) if it has a head

Right-Arc($label$) $(Stack|i, j|Buf, A) \rightarrow (Stack|i|j, Buf, A \cup \{(i, j, l)\})$: create edge $(i, j, label)$ between top of the stack (i) and next in buffer (j) , push j

Transition system

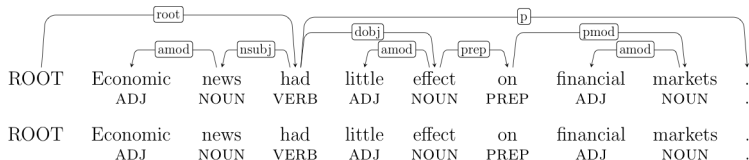
Shift $(Stack, i|Buf, A) \rightarrow (Stack|i, Buf, A)$: push next word from the buffer (i) to stack

Reduce $(Stack|i, Buf, A) \rightarrow (Stack, Buf, A)$: pop word top of the stack (i) if it has a head

Right-Arc($label$) $(Stack|i, j|Buf, A) \rightarrow (Stack|i|j, Buf, A \cup \{(i, j, l)\})$: create edge $(i, j, label)$ between top of the stack (i) and next in buffer (j), push j

Left-Arc($label$) $(Stack|i, j|Buf, A) \rightarrow (Stack, j|Buf, A \cup \{(j, i, l)\})$: create edge $(j, i, label)$ and pop i , if i has no head

Example

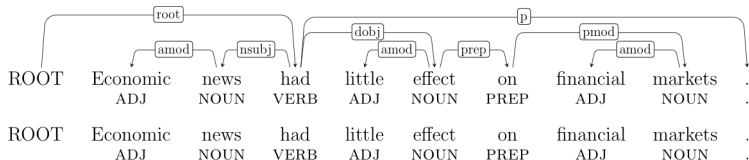


Stack = []

Buffer = [ROOT, Economic, news, had, little, effect, on, financial, markets, .]

Action?

Example

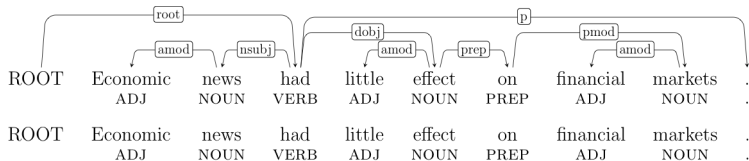


Stack = []

Buffer = [ROOT, Economic, news, had, little, effect, on, financial, markets, .]

Action? Shift

Example

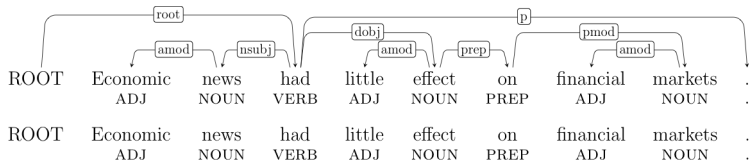


Stack = [ROOT]

Buffer = [Economic, news, had, little, effect, on, financial, markets, .]

Action?

Example

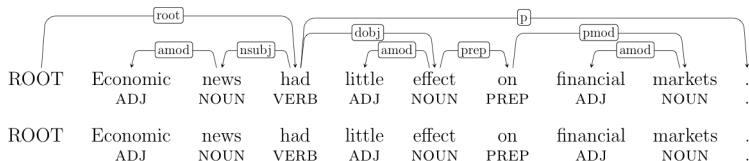


Stack = [ROOT]

Buffer = [Economic, news, had, little, effect, on, financial, markets, .]

Action? Shift

Example

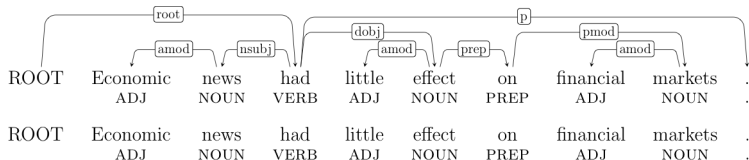


Stack = [ROOT, Economic]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action?

Example

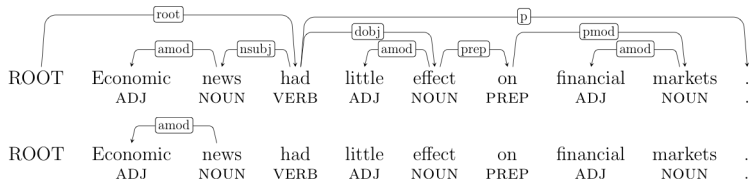


Stack = [ROOT, Economic]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action? Left-Arc(*amod*)

Example

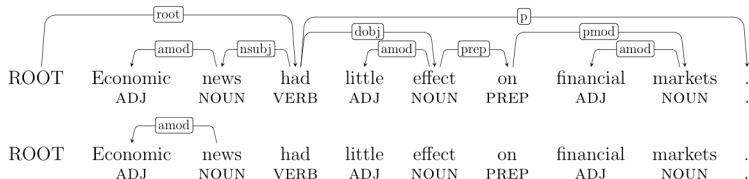


Stack = [ROOT]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action?

Example

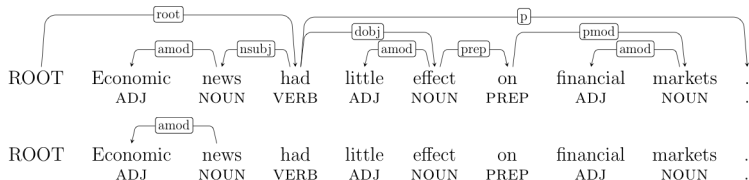


Stack = [ROOT]

Buffer = [news, had, little, effect, on, financial, markets, .]

Action? Shift

Example

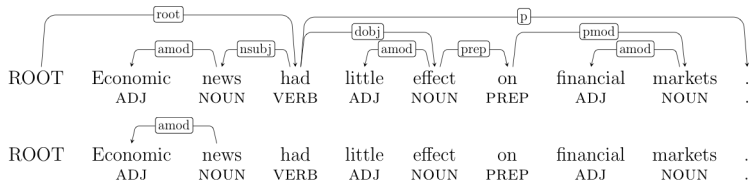


Stack = [ROOT, news]

Buffer = [had, little, effect, on, financial, markets, .]

Action?

Example

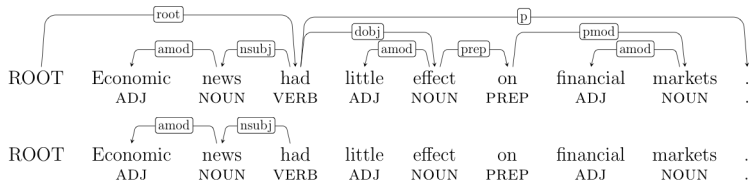


Stack = [ROOT, news]

Buffer = [had, little, effect, on, financial, markets, .]

Action? Left-Arc(*nsubj*)

Example

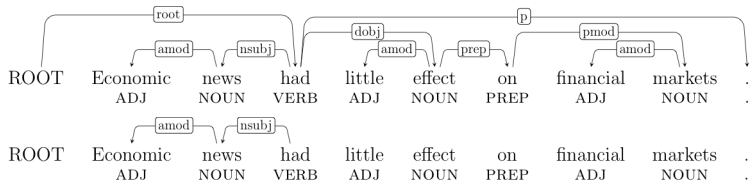


Stack = [ROOT]

Buffer = [had, little, effect, on, financial, markets, .]

Action?

Example

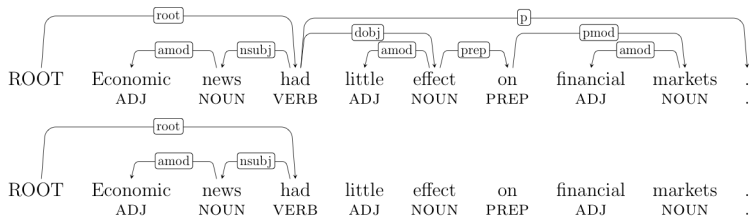


Stack = [ROOT]

Buffer = [had, little, effect, on, financial, markets, .]

Action? Right-Arc(*root*)

Example

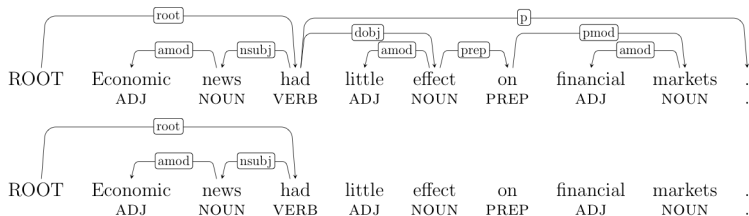


Stack = [ROOT, had]

Buffer = [little, effect, on, financial, markets, .]

Action?

Example

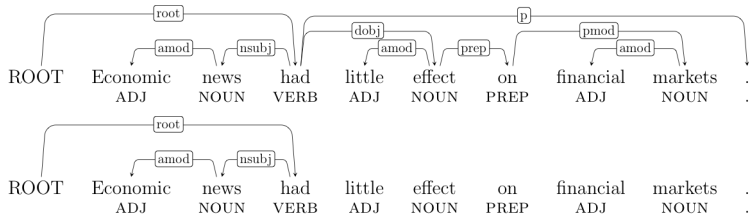


Stack = [ROOT, had]

Buffer = [little, effect, on, financial, markets, .]

Action? Shift

Example

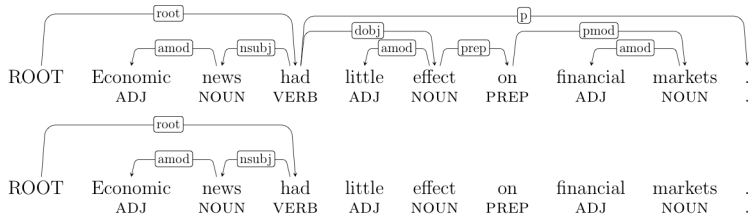


Stack = [ROOT, had, little]

Buffer = [effect, on, financial, markets, .]

Action?

Example

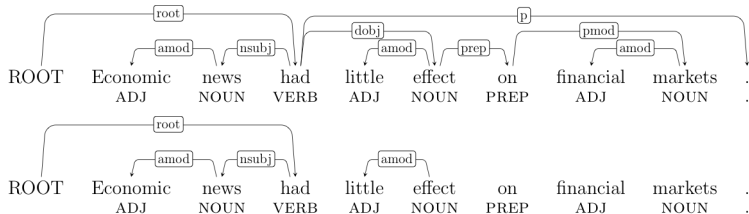


Stack = [ROOT, had, little]

Buffer = [effect, on, financial, markets, .]

Action? Left-Arc(*amod*)

Example

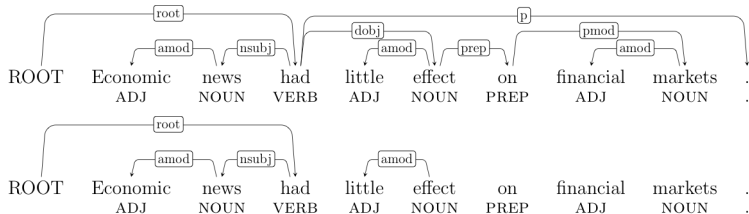


Stack = [ROOT, had]

Buffer = [effect, on, financial, markets, .]

Action?

Example

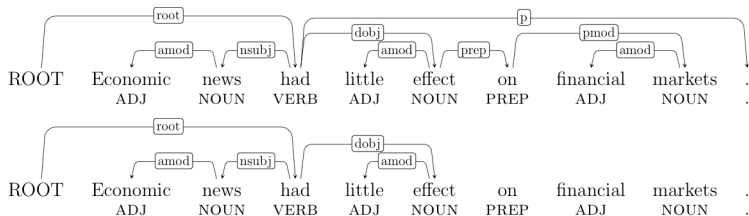


Stack = [ROOT, had]

Buffer = [effect, on, financial, markets, .]

Action? Right-Arc(*dobj*)

Example

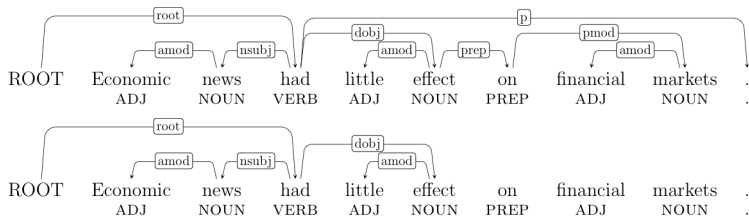


Stack = [ROOT, had, effect]

Buffer = [on, financial, markets, .]

Action?

Example

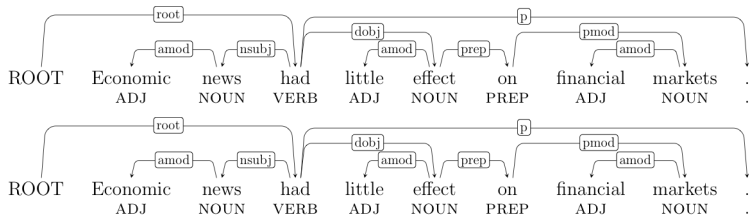


Stack = [ROOT, had, effect]

Buffer = [on, financial, markets, .]

Action? let's fast-forward...

Example

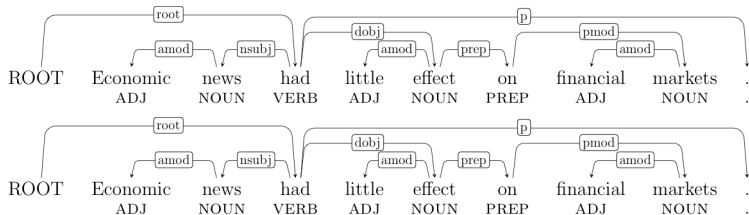


Stack = [ROOT, had, .]

Buffer = []

Empty buffer.

Example



Stack = [ROOT, had, .]

Buffer = []

Empty buffer. DONE!

Other transition systems?

- This was the arc-eager system. Others:

Other transition systems?

- This was the arc-eager system. Others:
 - arc-standard (3 actions)

Other transition systems?

- This was the arc-eager system. Others:
 - arc-standard (3 actions)
 - easy-first (not left-to-right), etc.
- All operate with actions combining:
 - moving words from the buffer to the stack and back (shift/un-shift)
 - popping words from the stack (reduce)
 - creating labeled arcs left and right

Other transition systems?

- This was the arc-eager system. Others:
 - arc-standard (3 actions)
 - easy-first (not left-to-right), etc.
- All operate with actions combining:
 - moving words from the buffer to the stack and back (shift/un-shift)
 - popping words from the stack (reduce)
 - creating labeled arcs left and right
- Intuition: Define actions that are easy to learn

Transition-based Dependency Parsing

Input: *sentence* \mathbf{x}

state $S_1 = \text{initialize}(\mathbf{x})$; *timestep* $t = 1$

while S_t not final **do**

action $\alpha_t = \arg \max_{\alpha \in \mathcal{A}} f(\alpha, S_t)$

$S_{t+1} = S_t(\alpha_t)$; $t = t + 1$

What is f ?

Transition-based Dependency Parsing

Input: *sentence* \mathbf{x}

state $S_1 = \text{initialize}(\mathbf{x})$; *timestep* $t = 1$

while S_t not final **do**

action $\alpha_t = \arg \max_{\alpha \in \mathcal{A}} f(\alpha, S_t)$

$S_{t+1} = S_t(\alpha_t)$; $t = t + 1$

What is f ? A multiclass classifier

What do we need to learn it?

Transition-based Dependency Parsing

Input: *sentence* \mathbf{x}

state $S_1 = \text{initialize}(\mathbf{x})$; *timestep* $t = 1$

while S_t not final **do**

action $\alpha_t = \arg \max_{\alpha \in \mathcal{A}} f(\alpha, S_t)$

$S_{t+1} = S_t(\alpha_t)$; $t = t + 1$

What is f ? A multiclass classifier

What do we need to learn it?

- learning algorithm (e.g. logistic regression)
- labelled training data
- feature representation

What are the right actions?



We only have sentences labelled with graphs:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

What are the right actions?



We only have sentences labelled with graphs:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

Ask an oracle to tell us the actions constructing the graph!

What are the right actions?



We only have sentences labelled with graphs:

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

Ask an oracle to tell us the actions constructing the graph!

In our case, a set of **rules** comparing the current state $S = (Stack, Buffer, ArcsPredicted)$ with G_x returning the correct action as label

Learning from an oracle

Given a labelled sentence and a transition system, an oracle returns states labelled with the correct actions.

$$D_{train} = \{(\mathbf{x}^1, G_x^1) \dots (\mathbf{x}^M, G_x^M)\}$$

$$\mathbf{x}^m = [x_1, \dots, x_N]$$

$$\text{graph } G_x = (V_x, A_x)$$

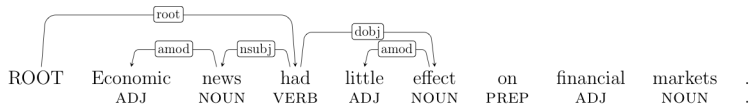
$$\text{vertices } V_x = \{0, 1, \dots, N\}$$

$$\text{edges } A_x = \{(i, j, k) | i, j \in V, k \in L(\text{labels})\}$$

$$\text{states } \mathbf{S}^m = [S_1, \dots, S_T]$$

$$\text{actions } \alpha^m = [\alpha_1, \dots, \alpha_T]$$

Feature Representation



Stack = [ROOT, had, effect]

Buffer = [on, financial, markets, .]

What features would help us predict the correction action
Right-Arc(*prep*)?

Feature Representation

- **Words/PoS in stack and buffer:**

wordS1=effect, wordB1=on, wordS2=had, posS1=NOUN,
etc.

Feature Representation

- **Words/PoS in stack and buffer:**

wordS1=effect, wordB1=on, wordS2=had, posS1=NOUN,
etc.

- **Dependencies so far:**

depS1=dobj, depLeftChildS1=amod,
depRightChildS1=NULL, etc.

Feature Representation

- **Words/PoS in stack and buffer:**

wordS1=effect, wordB1=on, wordS2=had, posS1=NOUN,
etc.

- **Dependencies so far:**

depS1=dobj, depLeftChildS1=amod,
depRightChildS1=NULL, etc.

- **Previous actions:**

$\alpha_{t-1} = \text{Right-Arc}(\text{dobj})$, etc.

Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones

Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies

Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies
- Graph-based lacks the rich structural features

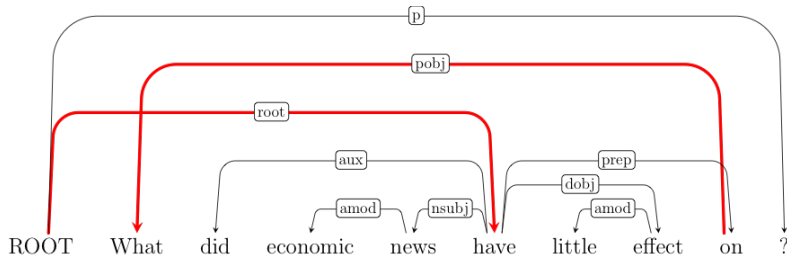
Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies
- Graph-based lacks the rich structural features
- Transition-based is greedy and suffers from early mistakes

Transition-based vs Graph-based parsing

- Transition-based tends to be better on shorter sentences, graph-based on longer ones
- Graph-based tends to be better on long-range dependencies
- Graph-based lacks the rich structural features
- Transition-based is greedy and suffers from early mistakes
- Actually, can we ameliorate the greedy issue?
Use Beam Search!

Non-Projectivity



- Arcs are crossing each other
- long-range dependencies
- free word order

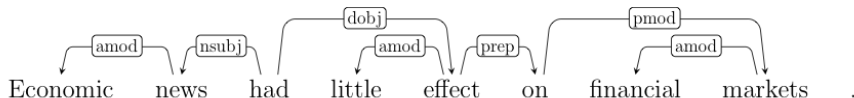
Non-projective Transition-based parsing

- The standard stack-based systems cannot do it.
- But there are extensions:
 - swap actions: word reordering
 - k-planar parsing: use multiple stacks (usually 2)
- Standard graph-based parsing handles non-projectivity.

Incremental Language Processing

- Other problems solved with similar approaches (a.k.a. transition-based, greedy):
 - semantic parsing (converting a natural language utterance to a logical form)
 - coreference resolution
- Whenever you have a problem with a very large space of outputs, worth considering

Evaluation



- **Head-finding word-accuracy:**

- unlabelled: % of words with the right head
- labelled: % of words with the right head and label

- **Sentence accuracy:** % of sentences with correct graph

Bibliography

- [Chapter 11](#) from Eisenstein
- Nivre and McDonald's tutorial [slides](#)
- Nivre's [article](#) on deterministic transition-based dependency parsing
- Nivre and McDonald's [paper](#) comparing their approaches

Coming up next week...

- Feed-forward Neural Networks
- Getting ready for Assignment 2!